The Great Type Hope

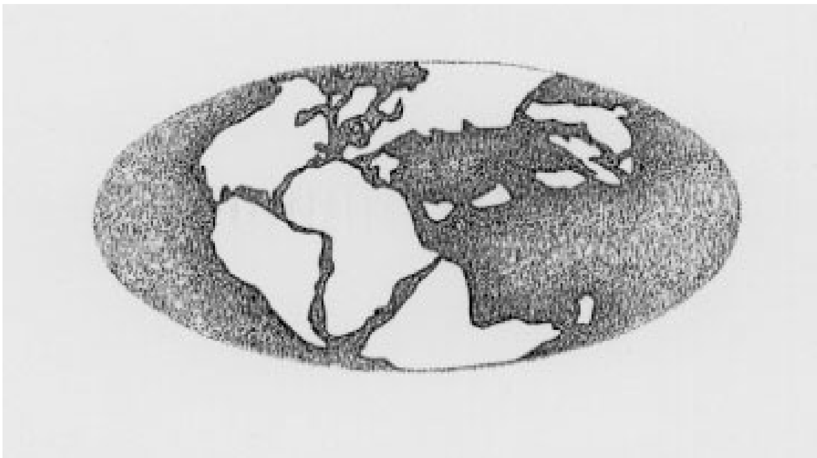
Philip Wadler, Avaya Labs

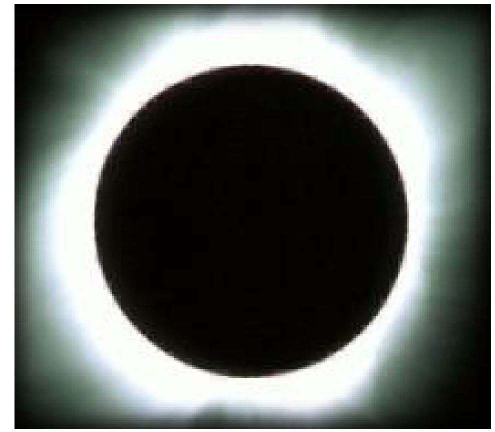
`wadler@avaya.com`

Part I

A logical coincidence

Coincidences





Coincidences

Curry-Howard

Hindley-Milner

Girard-Reynolds

Simply typed lambda calculus

$$\frac{}{A_1, \ldots, A_n \vdash A_i}\ \text{Id}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to\text{-I} \qquad \frac{\Gamma \vdash A \to B \qquad \Gamma \vdash A}{\Gamma \vdash B}\ \to\text{-E}$$

Simply typed lambda calculus

$$\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i} \text{Id}$$

$$\frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x^A.\, u : A \to B} \to\text{-I} \qquad \frac{\Gamma \vdash s : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B} \to\text{-E}$$

Polymorphic lambda calculus

$$\frac{}{A_1, \ldots, A_n \vdash A_i}\ \text{Id}$$

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to\text{-I} \qquad \frac{\Gamma \vdash A \to B \quad \Gamma \vdash A}{\Gamma \vdash B}\ \to\text{-E}$$

$$\frac{\Gamma \vdash B}{\Gamma \vdash \forall X.\, B}\ \forall^2\text{-I} \quad (X \text{ not free in } \Gamma) \qquad \frac{\Gamma \vdash \forall X.\, B}{\Gamma \vdash B[X := A]}\ \forall^2\text{-E}$$

Polymorphic lambda calculus

$$\frac{}{x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i}\ \text{Id}$$

$$\frac{\Gamma, x : A \vdash u : B}{\Gamma \vdash \lambda x^A.\, u : A \to B}\ \to\text{-I} \qquad \frac{\Gamma \vdash s : A \to B \qquad \Gamma \vdash t : A}{\Gamma \vdash s\, t : B}\ \to\text{-E}$$

$$\frac{\Gamma \vdash u : B}{\Gamma \vdash \Lambda X.\, u : \forall X.\, B}\ \forall^2\text{-I} \quad (X \text{ not free in } \Gamma) \qquad \frac{\Gamma \vdash s : \forall X.\, B}{\Gamma \vdash s\, A : B[X := A]}\ \forall^2\text{-E}$$

The Church numeral one

$$
\dfrac{\dfrac{\dfrac{\dfrac{\Gamma \vdash X \to X \qquad \Gamma \vdash X}{X \to X,\ X \vdash X}\ \to\text{-E}}{X \to X \vdash X \to X}\ \to\text{-I}}{\vdash (X \to X) \to X \to X}\ \to\text{-I}}{\vdash \forall X.\,(X \to X) \to X \to X}\ \forall^2\text{-I}
$$

$$
\Gamma \equiv X \to X,\ X
$$

The Church numeral one

$$\cfrac{\cfrac{\cfrac{\cfrac{\Gamma \vdash s : X \to X \qquad \Gamma \vdash z : X}{s : X \to X,\ z : X \vdash s\ z : X}\ \to\text{-E}}{s : X \to X \vdash \lambda z^{X}.\, s\ z : X \to X}\ \to\text{-I}}{\vdash \lambda s^{X \to X}.\, \lambda z^{X}.\, s\ z : (X \to X) \to X \to X}\ \to\text{-I}}{\vdash \Lambda X.\, \lambda s^{X \to X}.\, \lambda z^{X}.\, s\ z : \forall X.\, (X \to X) \to X \to X}\ \forall^2\text{-I}$$

$$\Gamma \ \equiv\ s : X \to X,\ z : X$$

Products

$$\frac{\Gamma \vdash A \qquad \Gamma \vdash B}{\Gamma \vdash A \times B} \times\text{-I}$$

$$\frac{\Gamma \vdash A \times B}{\Gamma \vdash A} \times\text{-E} \qquad \frac{\Gamma \vdash A \times B}{\Gamma \vdash B}$$

Products

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B} \times\text{-I}$$

$$\frac{\Gamma \vdash s : A \times B}{\Gamma \vdash \mathrm{fst}\ s : A} \times\text{-E} \qquad \frac{\Gamma \vdash s : A \times B}{\Gamma \vdash \mathrm{snd}\ s : B}$$

Products

$$\frac{\Gamma \vdash t : A \qquad \Gamma \vdash u : B}{\Gamma \vdash (t, u) : A \times B} \times\text{-I}$$

$$\frac{\Gamma \vdash s : A \times B}{\Gamma \vdash \mathrm{fst}\ s : A} \times\text{-E} \qquad \frac{\Gamma \vdash s : A \times B}{\Gamma \vdash \mathrm{snd}\ s : B}$$

$$\begin{array}{lcl}
A \times B & \equiv & \forall X.\, (A \to B \to X) \to X \\
(t, u) & \equiv & \Lambda X.\, \lambda k^{A \to B \to X}.\, k\ t\ u \\
\mathrm{fst}\ s & \equiv & s\ A\ (\lambda x^{A}.\, \lambda y^{B}.\, x) \\
\mathrm{snd}\ s & \equiv & s\ A\ (\lambda x^{A}.\, \lambda y^{B}.\, x)
\end{array}$$

Sums

$$\frac{\Gamma \vdash A}{\Gamma \vdash A + B}\ \text{+-I} \qquad \frac{\Gamma \vdash B}{\Gamma \vdash A + B}$$

$$\frac{\Gamma \vdash A + B \qquad \Gamma, A \vdash C \qquad \Gamma \vdash B \vdash C}{\Gamma \vdash C}\ \text{+-E}$$

Sums

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathrm{inl}\ t : A + B}\ \text{+-I} \qquad \frac{\Gamma \vdash u : B}{\Gamma \vdash \mathrm{inr}\ u : A + B}$$

$$\frac{\Gamma \vdash s : A + B \qquad \Gamma, x : A \vdash t : C \qquad \Gamma \vdash y : B \vdash u : C}{\Gamma \vdash \mathrm{case}\ t\ \mathrm{of}\ \mathrm{inl}\ x \longrightarrow u;\ \mathrm{inr}\ y \longrightarrow v : C}\ \text{+-E}$$

Sums

$$\frac{\Gamma \vdash t : A}{\Gamma \vdash \mathrm{inl}\ t : A + B}\ \text{+-I} \qquad \frac{\Gamma \vdash u : B}{\Gamma \vdash \mathrm{inr}\ u : A + B}$$

$$\frac{\Gamma \vdash s : A + B \qquad \Gamma, x : A \vdash t : C \qquad \Gamma \vdash y : B \vdash u : C}{\Gamma \vdash \mathrm{case}\ t\ \mathrm{of}\ \mathrm{inl}\ x \to u;\ \mathrm{inr}\ y \to v : C}\ \text{+-E}$$

$$\begin{array}{lcl} A + B & \equiv & \forall X.\, (A \to X) \to (B \to X) \to X \\ \mathrm{inl}\ t & \equiv & \Lambda X.\, \lambda j^{A \to X}.\, \lambda k^{B \to X}.\, j\ t \\ \mathrm{inr}\ u & \equiv & \Lambda X.\, \lambda j^{A \to X}.\, \lambda k^{B \to X}.\, k\ u \\ \mathrm{case}\ s\ \mathrm{of}\ \mathrm{inl}\ x \to t;\ \mathrm{inr}\ y \to u & \equiv & s\, C\, (\lambda x^{A}.\, t)\, (\lambda y^{B}.\, u) \end{array}$$

The Triumph of Type

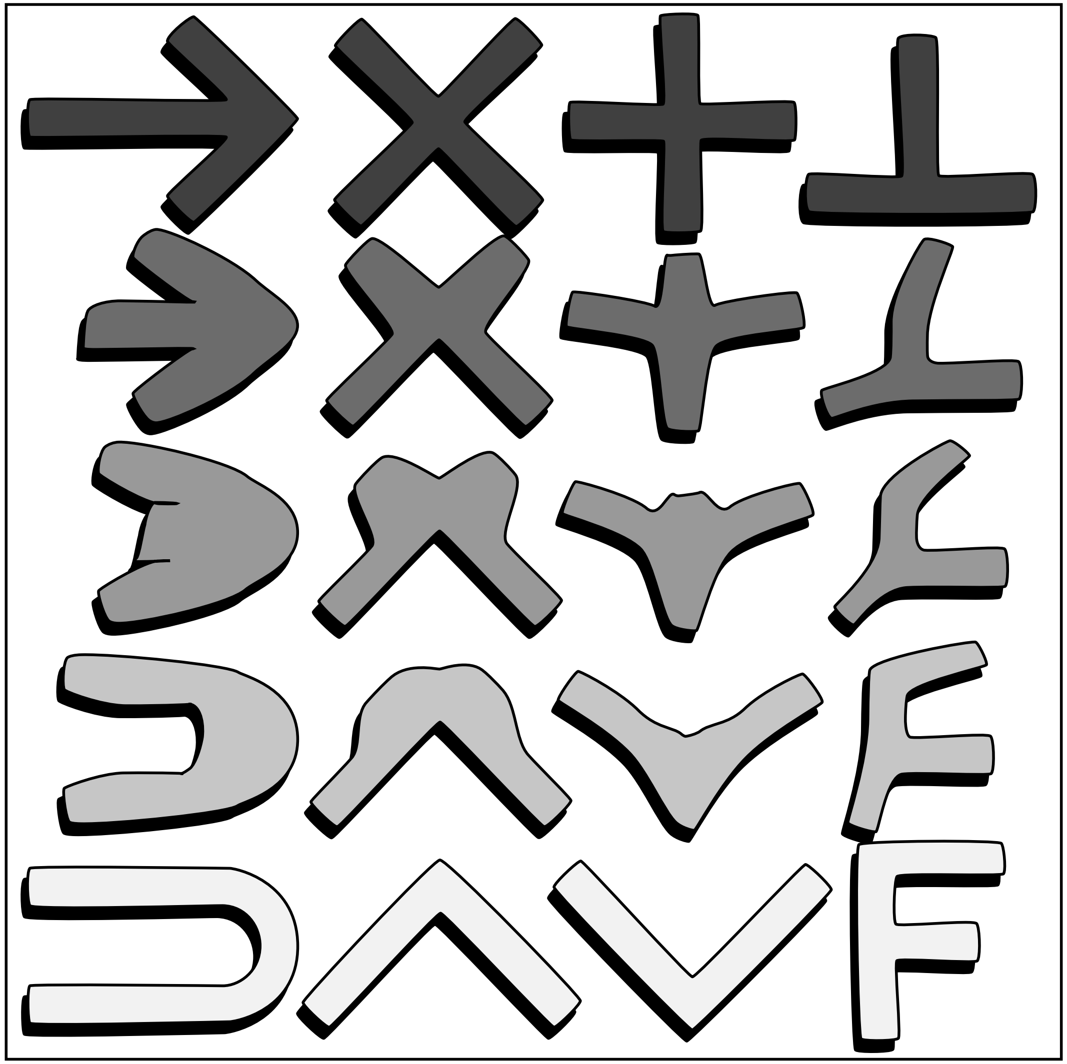
ML

Haskell

Java

XML/XQuery

Erlang?



LC'90 *The Curry-Howard homeomorphism*

Part II

Typed Erlang

Typed Erlang

```
-deftype tree(A,B) =
    T when T = empty | {branch,A,B,T,T}.

-type new() -> tree(0,0).
new() -> empty.
```

Inferred type

```
new() -> A when empty <= A
```

Simplified type

```
new() -> empty
```

Typed Erlang

```
-type insert(A,B,tree(A,B)) -> tree(A,B).
insert(K0,V0,empty) ->
    {branch,K0,V0,empty,empty};
insert(K0,V0,{branch,K,V,L,R}) ->
    if  K0 <  K ->
           {branch,K,V,insert(K0,V0,L),R};
        K0 == K ->
           {branch,K0,V0,L,R};
        true    ->
           {branch,K,V,L,insert(K0,V0,R)}
    end.
```

Inferred type

```
insert(B, C, D) -> A
when
branchE,F,G,A <= A; branchB,C,G,H <= A;
branchE,F,A,H <= A;
branchB,C,empty,empty <= A;
D <= empty | branchE,F,G,H;
G <= empty | branchE,F,G,H;
H <= empty | branchE,F,G,H;
H <= D; G <= D.
```

Simplified type

```
insert(D, E, F) -> A
when
empty | branchD,E,A,A <= A;
F <= empty | branchD,E,F,F.
```

Typed Erlang

```
-type lookup(A,tree(A,B)) -> B | error
        when B \ error.
lookup(K0,empty) -> error;
lookup(K0,{branch,K,V,L,R}) ->
    if  K0 <  K -> lookup(K0,L);
        K0 == K -> V;
        true    -> lookup(K0,R)
    end.
```

Inferred type

```
lookup(B, C) -> A
when
error <= A
C <= empty | branchD,E,F,G;
F <= empty | branchD,E,F,G;
G <= empty | branchD,E,F,G;
E <= A; F <= C; G <= C.
```

Simplified type

```
lookup(1, B) -> error | A
when
B <= empty | branch1, error | A, B, B;
A  error.
```

Part III

Details

Syntax

f, g			function names
c, d			constructors
X, Y, Z			variables
E	$::=$	X	expression
	$\mid$	$f(\overline{E})$	
	$\mid$	$c\{\overline{E}\}$	
	$\mid$	$\textbf{case}\ E_0\ \textbf{of}\ c_1\{\overline{X_1}\} \rightarrow E_1; \cdots; c_n\{\overline{X_n}\} \rightarrow E_n; X \rightarrow E_{n+1}$	
$prog$	$::=$	$f_1(\overline{X_1}) \rightarrow E_1; \cdots; f_n(\overline{X_n}) \rightarrow E_n$	program

Types

c, d			constructors
α, β			type variables
U, V	$::=$	$P \mid U$	union type
	$\mid$	R	
P, Q	$::=$	$c\{\overline{U}\}$	prime type
R	$::=$	α^{cs}	remainder
	$\mid$	1^{cs}	
	$\mid$	0	

Typing rules

$$\frac{}{F;\ A, X : U;\ C \vdash X : U} \quad \text{(Var)}$$

$$\frac{F;\ A;\ C \vdash E : U \qquad C \Vdash U \subseteq V}{F;\ A;\ C \vdash E : V} \quad \text{(Sub)}$$

$$\frac{F;\ A;\ C \vdash E_1 : U_1 \quad \ldots \quad F;\ A;\ C \vdash E_n : U_n}{F;\ A;\ C \vdash \overline{E} : \overline{U}} \quad \text{(Multi)}$$

Typing rules

$$\frac{}{F, f : \forall\overline{\alpha}.(\overline{U}) \to V \ \textbf{when}\ D;\ A;\ C, D[\overline{V}/\overline{\alpha}] \vdash f : ((\overline{U}) \to V)[\overline{V}/\overline{\alpha}]} \quad (\textsc{Fun})$$

$$\frac{F;\ A;\ C \vdash f : (\overline{U}) \to V \qquad F;\ A;\ C \vdash \overline{E} : \overline{U}}{F;\ A;\ C \vdash f(\overline{E}) : V} \quad (\textsc{Call})$$

$$\frac{\begin{array}{c} F, f : ((\overline{U}) \to V \ \textbf{when}\ C);\ \overline{X} : \overline{U};\ C \vdash E : V \\ \mathrm{FTV}((\overline{U}) \to V \ \textbf{when}\ C) = \overline{\alpha} \end{array}}{F;\ \emptyset;\ C \vdash f(\overline{X}) \to E : (\forall\overline{\alpha}.(\overline{U}) \to V \ \textbf{when}\ C)} \quad (\textsc{Def})$$

Typing rules

$$\frac{F;\ A;\ C \vdash \overline{E} : \overline{U}}{F;\ A;\ C \vdash c\{\overline{E}\} : c\{\overline{U}\}} \quad (\textsc{Con})$$

$$\frac{\begin{array}{c} F;\ A;\ C \vdash E_0 : c_1\{\overline{U_1}\} \mid \ldots \mid c_n\{\overline{U_n}\} \mid U \\ F;\ A, \overline{X_1} : \overline{U_1};\ C \vdash E_1 : V \quad \ldots \quad F;\ A, \overline{X_n} : \overline{U_n};\ C \vdash E_n : V \\ F;\ A, X : U;\ C \vdash E_{n+1} : V \end{array}}{F;\ A;\ C \vdash (\textbf{case}\ E_0\ \textbf{of}\ c_1\{\overline{X_1}\} \to E_1; \ldots c_n\{\overline{X_n}\} \to E_n; X \to E_{n+1}\ \textbf{end})\ : V} \quad (\textsc{Case})$$

Constraint reduction

$$
\begin{array}{llll}
P \mid U \subseteq V & \Rightarrow & P \subseteq V, U \subseteq V & \\
0 \subseteq U & \Rightarrow & \text{none} & \\
1^{cs} \subseteq 0 & \Rightarrow & \text{fail} & \\
1^{cs} \subseteq c\{\overline{U}\} \mid U & \Rightarrow & 1 \subseteq \overline{U},\ 1^{cs} \subseteq U & \text{if } c \notin cs \\
 & & 1^{cs} \subseteq U & \text{otherwise} \\
1^{cs} \subseteq 1^{ds} & \Rightarrow & \text{none} & \text{if } ds \subseteq cs \\
 & & \text{fail} & \text{otherwise} \\
1^{cs} \subseteq \alpha^{ds} & \Rightarrow & 1^{cs} \subseteq \alpha^{ds} & \text{if } ds \subseteq cs \\
 & & \text{fail} & \text{otherwise}
\end{array}
$$

Constraint reduction

$$
\begin{array}{lcll}
c\{\overline{U}\} \subseteq 0 & \Rightarrow & \text{fail} & \\
c\{\overline{U}\} \subseteq c'\{\overline{U}'\} \mid U & \Rightarrow & \overline{U} \subseteq \overline{U}' & \text{if } c = c' \\
 & & c\{\overline{U}\} \subseteq U & \text{otherwise} \\
c\{\overline{U}\} \subseteq 1^{cs} & \Rightarrow & \text{none} & \text{if } c \notin cs \\
 & & \text{fail} & \text{otherwise} \\
c\{\overline{U}\} \subseteq \alpha^{cs} & \Rightarrow & c\{\overline{U}\} \subseteq \alpha^{cs} & \text{if } c \notin cs \\
 & & \text{fail} & \text{otherwise} \\
U \subseteq \alpha^{cs},\ \alpha^{cs} \subseteq V & \Rightarrow & \multicolumn{2}{l}{U \subseteq V, U \subseteq \alpha^{cs}, \alpha^{cs} \subseteq V}
\end{array}
$$

Part IV

A fly in the ointment

And

```
-datatype bool() = true | false.

-type and(bool(),bool()) -> bool().
and(true,true) -> true;
and(false,X)   -> false;
and(X,false)   -> false.
```

Uh oh

```
-type and(1,false) -> false | true.
and(X,Y) ->
   let Z = (case Y of false -> false end) in
   case X of
      true  ->
         case Y of
            true -> true;
            X -> Z
         end;
      false -> false;
      X -> Z
  end.
```

Part V

A simpler approach?

Typed Erlang, simplified

```
-deftype tree(A,B) =
    empty | {branch,A,B,T,T}.

-type new() -> tree(A,B).
new() -> empty.
```

Typed Erlang

```
-type insert(A,B,tree(A,B)) -> tree(A,B).
insert(K0,V0,empty) ->
    {branch,K0,V0,empty,empty};
insert(K0,V0,{branch,K,V,L,R}) ->
    if  K0 <  K ->
           {branch,K,V,insert(K0,V0,L),R};
        K0 == K ->
           {branch,K0,V0,L,R};
        true    ->
           {branch,K,V,L,insert(K0,V0,R)}
    end.
```

Typed Erlang

```
-deftype sum(A,B) =
    inl(A) | inr(B).

-deftype error =
    error

-type lookup(A,tree(A,B)) -> inl(B) | inr(error)
lookup(K0,empty) -> inr(error);
lookup(K0,{branch,K,V,L,R}) ->
    if  K0 <  K -> lookup(K0,L);
        K0 == K -> inl(V);
        true    -> lookup(K0,R)
    end.
```

Part VI

A simpler but more powerful approach?

Types and logic

$$s \in A \rightarrow B$$

$$\equiv$$

$$\forall x.\, x \in A \rightarrow s\, x \in B$$

Retrofitting types

```
-type lookup(A,tree(A,B)) -> B | error
        when B \ error.
lookup(K0,empty) -> error;
lookup(K0,{branch,K,V,L,R}) ->
    if  K0 <  K -> lookup(K0,L);
        K0 == K -> V;
        true    -> lookup(K0,R)
    end.
```

Retrofitting types

```
-assert  K in A
      &  T in tree(A,B)
      &  V = lookup(K,T)
      &  not (error in B)
      -> V in B \/ V in error.
lookup(K0,empty) -> error;
lookup(K0,{branch,K,V,L,R}) ->
    if  K0 <  K -> lookup(K0,L);
        K0 == K -> V;
        true    -> lookup(K0,R)
    end.
```

Part VII

Conclusions

Conclusions

Types are good

Erlang is good

Typed Erlang could be better

Conclusions

Types are good

Erlang is good

Typed Erlang could be better

Long live λ calculus!

Further reading

Simon Marlow and Philip Wadler, A practical subtyping system for Erlang, *2'nd International Conference on Functional Programming*, Amsterdam, June 1997.

Philip Wadler, New Languages, Old Logic, *Dr Dobbs Journal*, special supplement on *Software in the 21st century*, December 2000. (See also, 19th century logic and 21st century computing, on my web page.)

Philip Wadler, The Girard-Reynolds isomorphism, *Theoretical Aspects of Computer Software* Sendai, Japan, October 2001. Journal version to appear in *Information and Computation*.